TP Optimisation

Compte rendu de TP

Francisco Italo GUEDES CARVALHO - italo.guedes@ecole.ensicaen.fr
Samuel Dovi - samuel.dovi@ecole.ensicaen.fr
Vítor Lima Aguirra - vitor.lima-aguirra@ecole.ensicaen.fr

Caen, 4 novembre 2024.

Introduction

Les objectifs du TP sont d'implémenter et d'utiliser les algorithmes d'optimisation vue en cours et en TD, afin de résoudre plusieurs problèmes d'optimisation. Ce compte rendu porte sur le TP 1 et le TP 2.

Pour la résolution des problèmes proposés, les algorithmes ont été implémentés en langage *Octave/Matlab*. Chaque algorithme est une développement du algorithme précédent, donc, il était possible d'implémenter une bibliothèque qu'ensemble toutes les algorithmes dans une seule archive, que se trouve dans l'appendice 2.2.

TP 1

On considère la fonction "*banane*" de Rosenbrocks définit par

$$f(x_1, x_2) = (x_1 - 0.5)^2 + p(x_1^2 - x_2)^2$$

où p = 10, On se propose alors d'utilisés quatre algorithmes différents afin de comparer les performances et la précision de chacun d'entre eux.

Algorithme du gradient à pas optimal

Pour cet algorithme, on se décide à calculer le gradient de la fonction de Rosenbrocks.

$$\nabla f(x_1, x_2) = \begin{bmatrix} 2(x_1 - 0.5) + 4p(x_1^2 - x_2) \cdot x_1 \\ 2p(x_1^2 - x_2) \end{bmatrix}$$

Cet algorithme nous permet de trouver la meilleure direction de descente pour trouver l'extremum voulu.

Pour réaliser l'optimisation, nous avons modélisé la fonction dans MATLAB, utilisée pour créer le graphique, ainsi que son gradient. Le contour de la fonction banane est montré dans la Figure 1.

L'algorithme du pas optimal consiste à minimiser une fonction en obtenant la direction de descente optimale, qui est le négatif du gradient, et en trouvant le pas optimal pour minimiser la fonction.

Il est intéressant d'obtenir le pas optimal car, si le pas utilisé est trop petit, l'optimisation prendra trop de temps pour converger, si le pas est trop grand, l'algorithme risque de ne pas converger.

Deux algorithmes peuvent être utilisés pour trouver le pas optimal : Aramijo et l'interpolation parabolique.

Les Figures 2 et 3 montrent l'optimisation de la fonction banane de Rosenbrocks avec l'algorithme de pas optimal, utilisant respectivement les méthodes d'Aramijo, à gauche, et d'interpolation parabolique à droite.

[H]

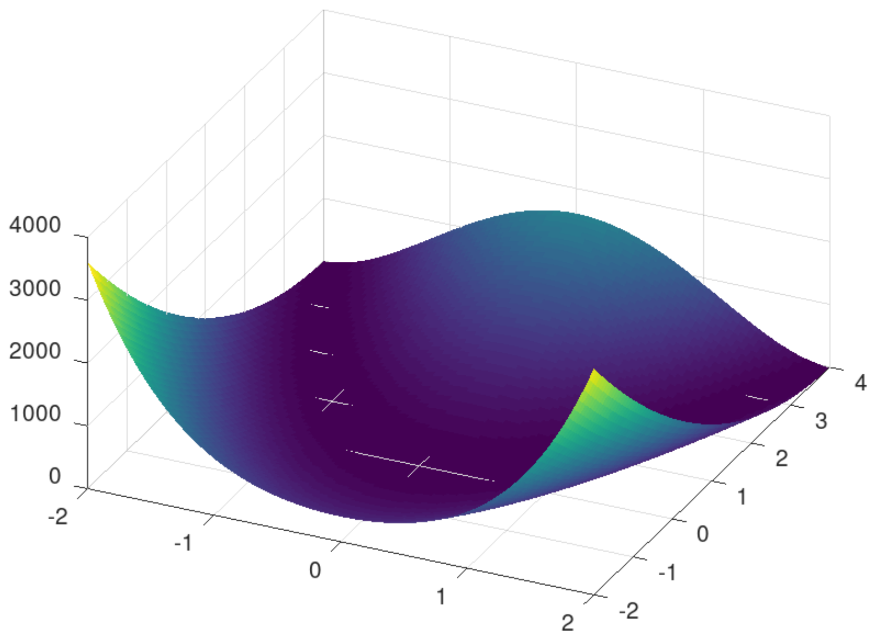

FIGURE 1 – Fonction Banane contour.

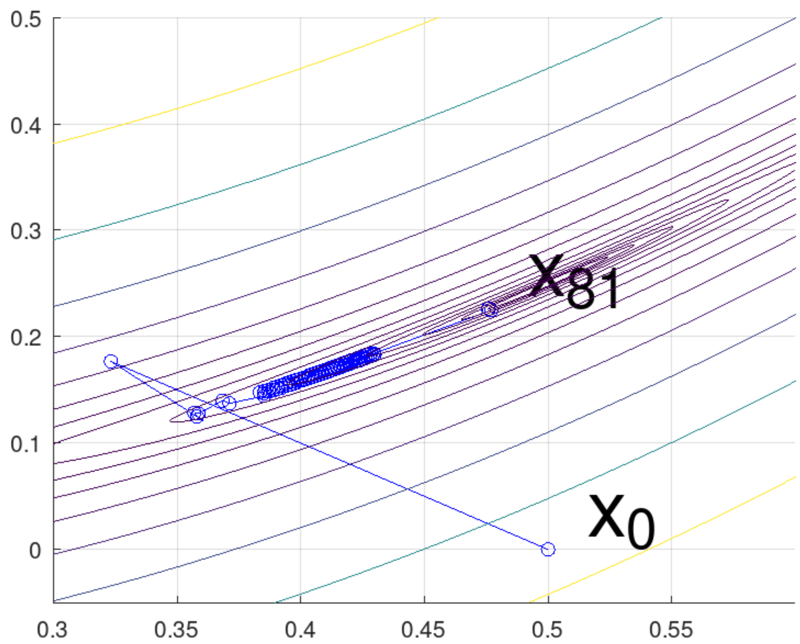

FIGURE 2 – Aramijo méthode.

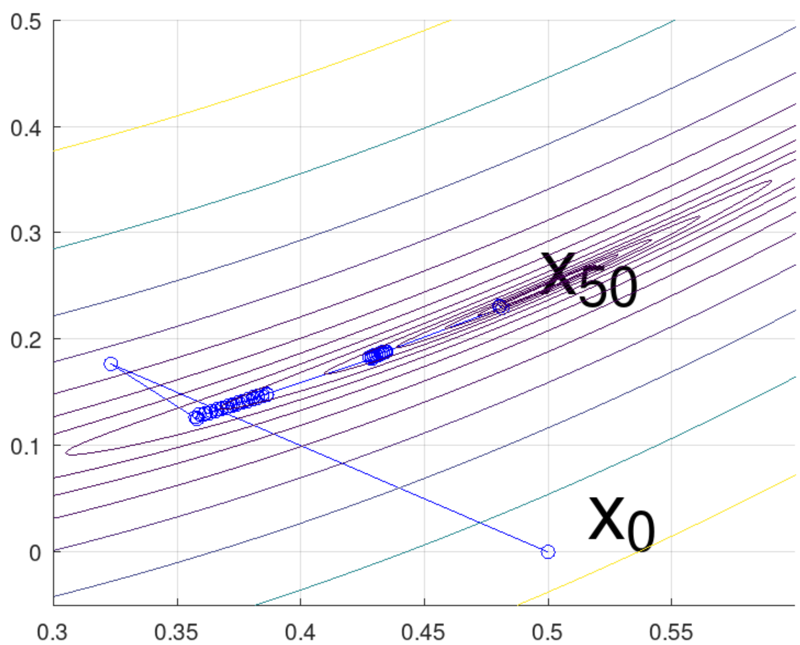

FIGURE 3 – Interpolation Parabolique.

Optimisation avec l'algorithme du gradient conjugué

L'algorithme du gradient conjugué est une amélioration par rapport à l'algorithme d'optimisation du pas optimal dans le sens où il cherche également à minimiser une fonction en obtenant la direction de descente optimale, qui est le négatif du gradient, puis en recherchant le pas optimal pour réaliser l'optimisation.

La différence ici est qu'un terme β est ajouté à la direction de descente de manière à ce que l'algorithme converge plus rapidement.

Deux algorithmes sont proposés pour trouver β : Fletcher & Reeves et Polak & Ribière.

Tout d'abord, nous avons réalisé l'optimisation par gradient conjugué en utilisant l'algorithme d'Aramijo pour trouver le pas optimal.

Les Figures 4 et 5 montrent l'optimisation par gradient conjugué avec les méthodes de Fletcher & Reeves, à gauche, et de Polak & Ribière, à droite. Les deux optimisations ont été réalisées avec l'algorithme d'Aramijo pour trouver le pas optimal.

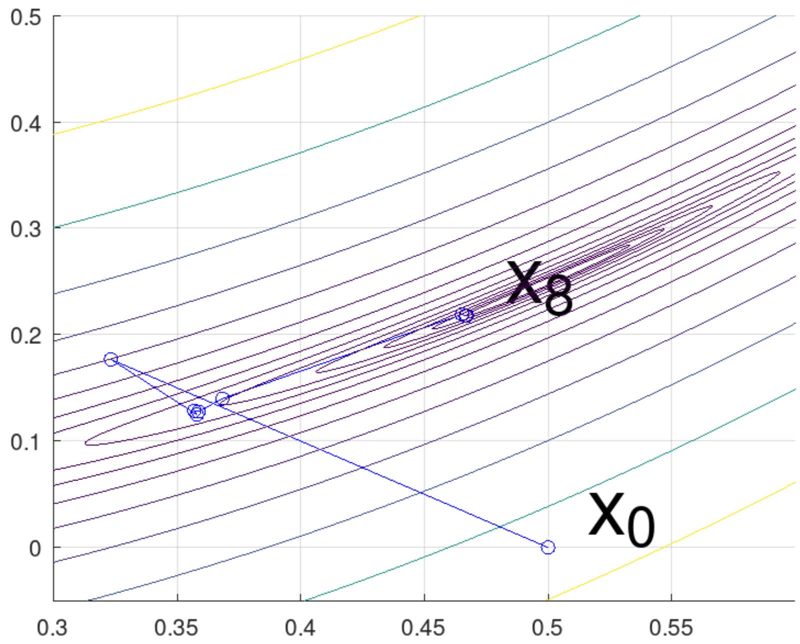

FIGURE 4 – Fletcher & Reeves avec Aramijo.

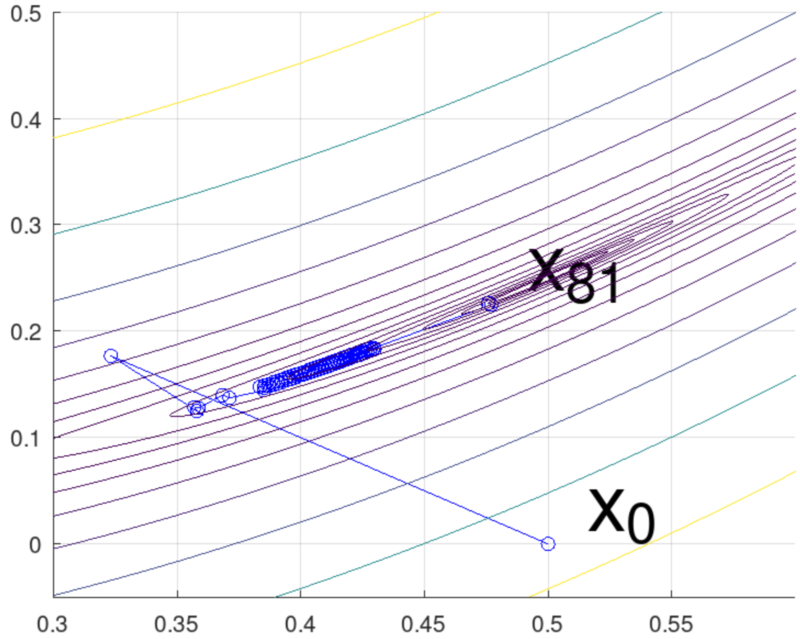

FIGURE 5 – Polak & Ribière méthode avec Aramijo.

Dans les Figures 6 et 7, les mêmes optimisations par gradient conjugué ont été réalisées, mais en utilisant l'algorithme d'interpolation parabolique pour trouver le pas optimal.

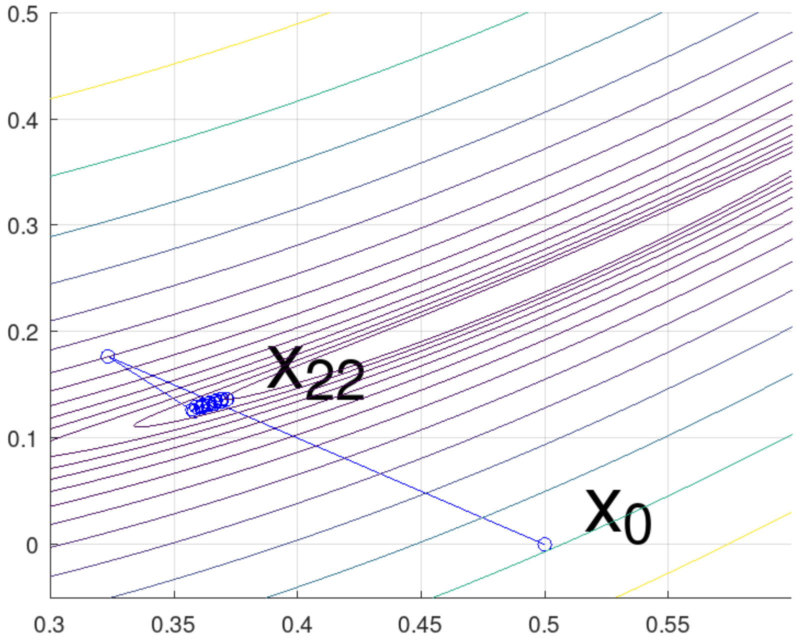

FIGURE 6 – Fletcher & Reeves avec l'Interpolation Parabolique.

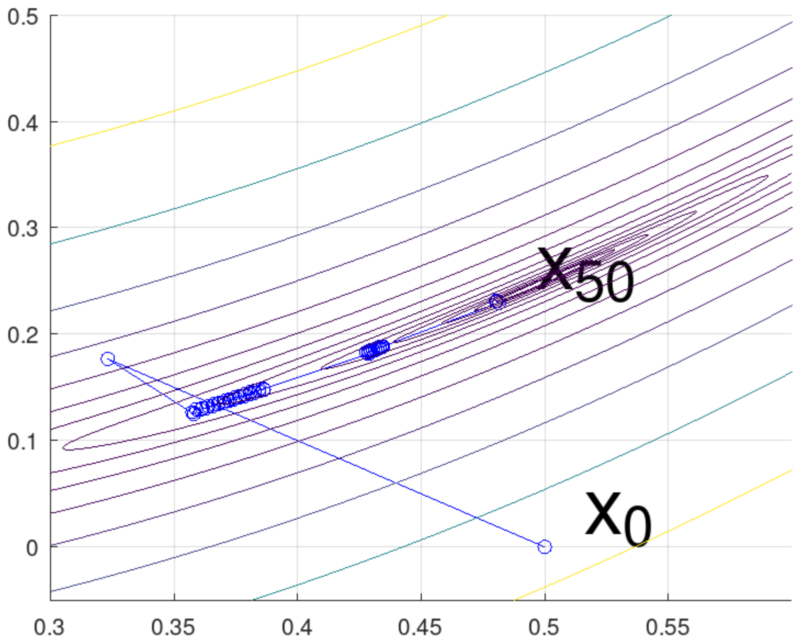

FIGURE 7 – Polak & Ribière méthode avec l'Interpolation Parabolique.

Finalement, les Figures 8 et 9 contient les résultats des optimisations avec les algorithmes de Quasi-Newton, *DPF* et *BFGS*, respectivement.

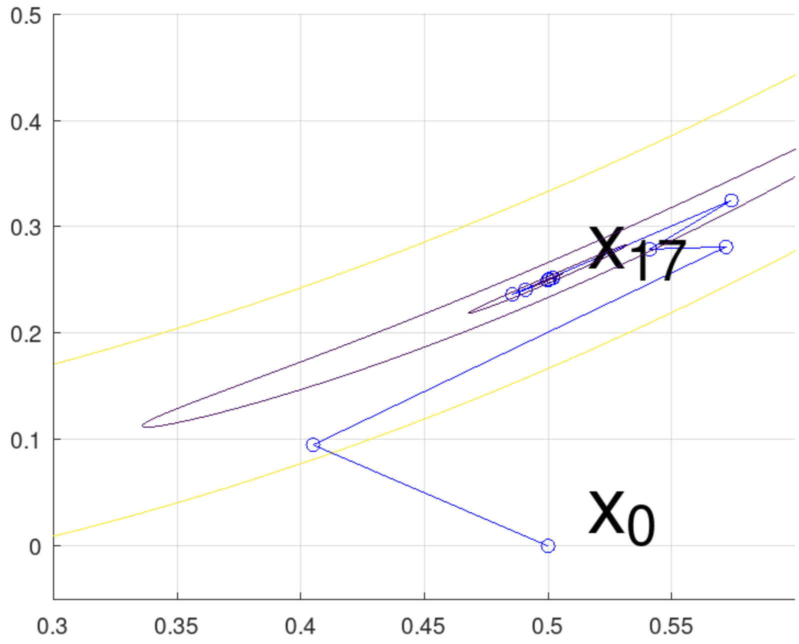

FIGURE 8 – *DPF* algorithme.

Le tableau 1 résume les résultats de chaque algorithme.

Méthode	Solution	Coût	Iterations
Armijo	[0.47595, 0.2263]	0.00058365	81
Parabolic	[0.48065, 0.23085]	0.0003775	50
Armijo, Fletcher	[0.46708, 0.21774]	0.0011016	8
Armijo, Ribière	[0.47595, 0.2263]	0.00058365	81
Parabolic, Fletcher	[0.37146, 0.13637]	0.016782	22
Parabolic, Ribière	[0.48065, 0.23085]	0.0003775	50
Parabolic, *DPF*	[0.5, 0.25]	3.1667e-19	17
Parabolic, *BFGS*	[0.4759, 0.20705]	0.038337	4

TABLE 1 – Résume des résultats du TP1

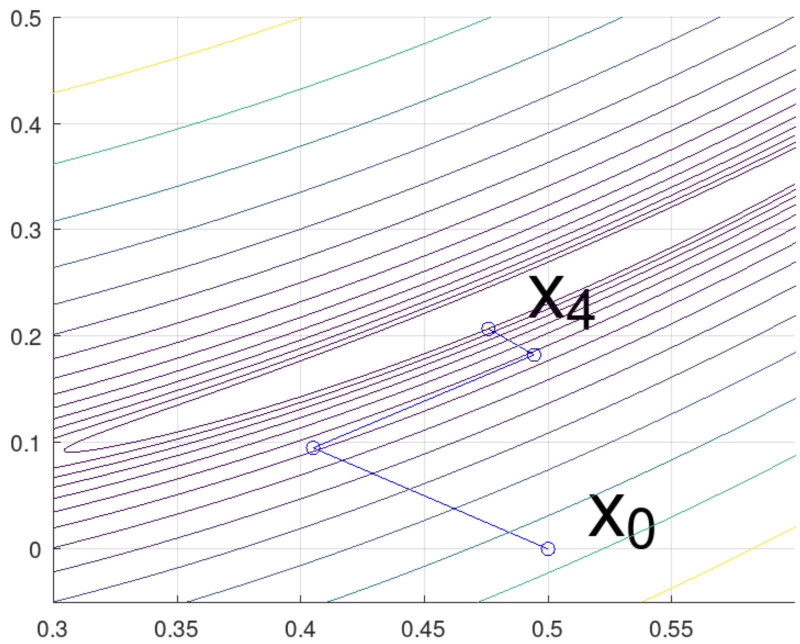

FIGURE 9 – *BFGS* algorithme.

TP 2

Penalization

Pour restreindre le domaine des solutions possibles à un problème de minimisation, une approche consiste à utiliser la pénalisation. La fonction est modifiée de manière à ce que sa sortie soit artificiellement élevée lorsque des arguments en dehors de la plage souhaitée sont fournis.

Lorsqu'une contrainte est ajoutée à un problème de minimisation, une nouvelle fonction, appelée fonction de pénalisation, est ajoutée à la fonction à minimiser. La pénalisation est nulle pour tous les arguments à l'intérieur du domaine souhaité et augmente progressivement à mesure que l'entrée s'éloigne de celui-ci. Généralement, la pénalisation est prise comme la distance au carré entre l'entrée et les limites du domaine acceptable. Un facteur d'échelle $1/\eta$ est ensuite utilisé pour augmenter la pénalisation afin que ses dimensions correspondent à celles de la fonction à minimiser, garantissant ainsi que le résultat ne sera pas trop en dehors du domaine souhaité et assurant la stabilité de l'algorithme.

Problème 0

Le Problème 0 consiste en la minimisation de la fonction donnée dans l'Équation 1, sous les contraintes indiquées dans l'Équation 2.

En utilisant la méthode de recherche de pas alpha d'Aramijo avec la méthode de recherche de beta de Fletcher, l'algorithme du gradient conjugué a convergé après 10 itérations sur $x_{min} = -0.0050001$ avec un minimum de $f_0(x_{min}) = 0.9975$.

$$\min f_0(x, y) \triangleq 1 + x + \frac{1}{3}x^3 \tag{1}$$

$$x \leq 0, x \in \mathcal{R} \tag{2}$$

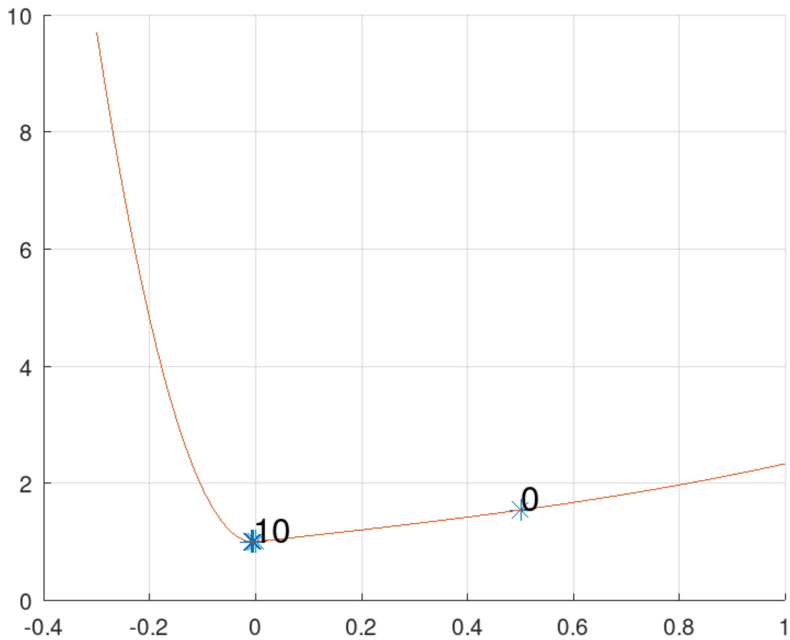

FIGURE 10 – Chemin de la solution d'optimisation du problème 0

Problème 1

Le Problème 1 consiste en la minimisation de la fonction donnée par l'Équation 3, sous les contraintes indiquées dans l'Équation 4.

En utilisant la méthode de recherche de pas alpha d'Aramijo avec la méthode de recherche de beta de Fletcher, l'algorithme du gradient conjugué a convergé après 9 itérations au point $x_{min} = \{-0.48309, -0.48295\}$ avec un minimum de $f_1(x_{min}) = 1.6908$.

$$\min f_1(x, y) \triangleq 2x^2 + 3xy + 2y^2 \tag{3}$$

$$x \leq -\frac{1}{2}, y \leq -\frac{1}{2}, (x, y) \in \mathcal{R}^2 \tag{4}$$

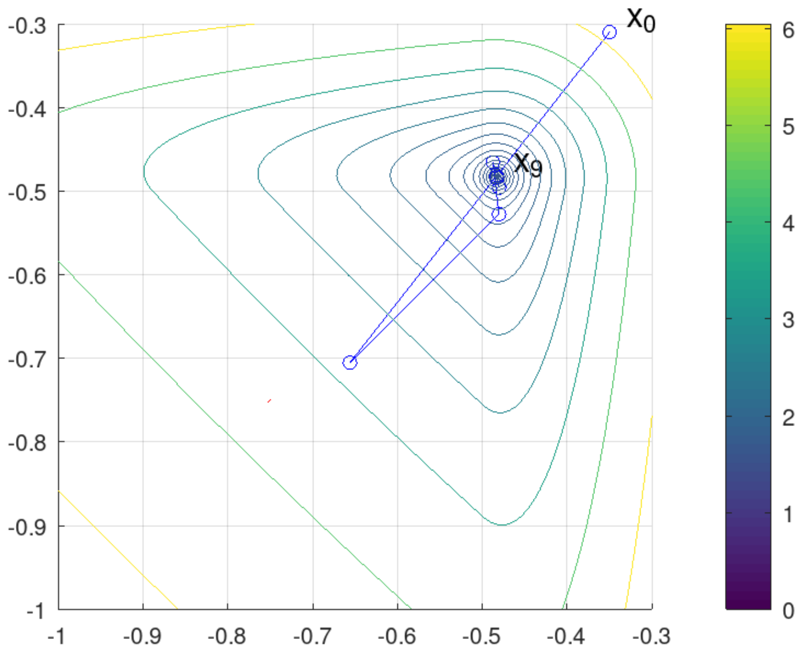

FIGURE 11 – Chemin de la solution d'optimisation du problème 1

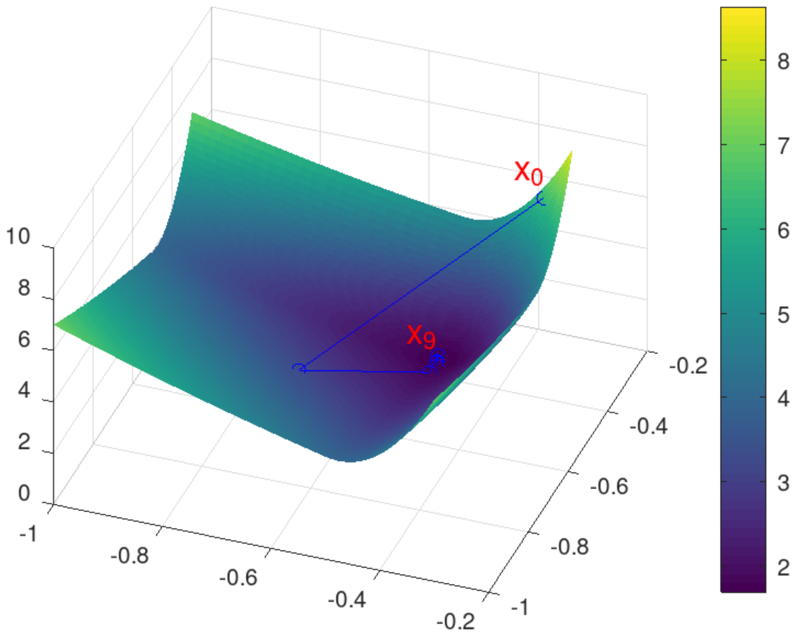

FIGURE 12 – Optimisation du problème 1

Problème 2

Le Problème 2 consiste en la minimisation de la fonction donnée dans l'Équation 5, sous les contraintes indiquées dans l'Équation 6.

En utilisant la méthode de recherche de pas alpha d'Aramijo avec la méthode de recherche de beta de Fletcher, l'algorithme du gradient conjugué a convergé après 10 itérations sur $x_{min} = 3.8379$ avec un minimum de $f_2(x_{min}) = 8.3156$.

$$\min f_2(x) \triangleq x^2 + 10\sin(x) \tag{5}$$

$$x \in [2, 10] \tag{6}$$

Problème 3

Le Problème 3 consiste en la minimisation de la fonction donnée dans l'Équation 7, sous les contraintes indiquées dans l'Équation 8.

En utilisant la méthode de recherche de pas alpha d'Aramijo avec la méthode de recherche de beta de Fletcher, l'algorithme du gradient conjugué

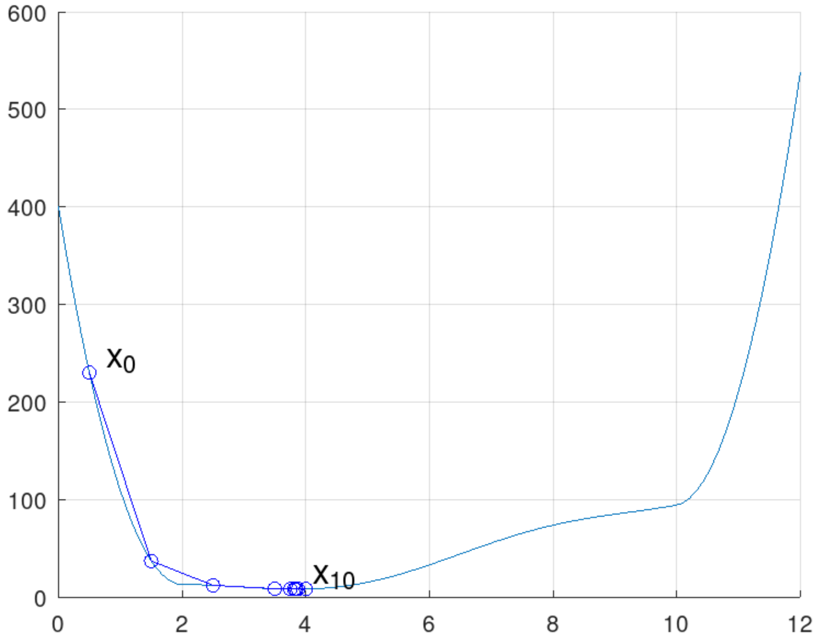

FIGURE 13 – Chemin de la solution d'optimisation du problème 2

a convergé après 331 itérations au point $x_{min} = \{-1.6244e-12, -1.0627e-12, -2.7142e-12\}$ avec un minimum de $f_3(x_{min}) = 7.0049e-23$.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\min f_3(x) \triangleq x^T A x, A = \begin{bmatrix} 5 & 3 & 1 \\ 3 & 2 & 1 \\ 1 & 1 & 4 \end{bmatrix} \tag{7}$$

$$x_1 + x_2 \leq x_3, 3x_1 - 2x_2 = x_3 \tag{8}$$

Problème 4

En considérant $n = 4$, $x_0 = [1\ 1\ 1\ 1]^T$, $A = [3\ 1\ 0\ 2]$, et $b = 1$, le Problème 4 consiste en la minimisation de la fonction donnée dans l'Équation 10, sous les contraintes indiquées dans l'Équation 11. La solution exacte est donnée dans l'Équation 9.

$$x^* = x_0 + A^T(AA^T)^{-1}(b - Ax_0) \tag{9}$$

En utilisant la méthode de recherche de pas alpha d'Aramijo avec la méthode de recherche de beta de Fletcher, l'algorithme du gradient conjugué a convergé après 376 itérations sur

$x_{min} = -0.067942, 0.63728, 0.99879, 0.28399$

avec un minimum de $f_4(x_{min}) = 0.89257$. La solution analytique du problème donne

$x^* = -0.071429, 0.64286, 1, 0.28571$

avec une distance minimale de $f_4^* = 0.89286$.

Le grand nombre de variables paramétrées dans ce problème rend l'évaluation de la qualité des résultats difficile. Par conséquent, afin d'évaluer les performances de l'algorithme, la distance entre les résultats numériques et analytiques a été déterminée, mesurant 0.0069066.

$$\min f_4(x) \triangleq \frac{1}{2}(x - x_0)^T (x - x_0) \tag{10}$$

$$Ax = b \tag{11}$$

1 Conclusion

Il est intéressant de noter que la plupart des modèles d'optimisation dérivent de la même équation de base 12.

$$x^{k+1} = x^k + \alpha d_k \tag{12}$$

Ces modifications entraînent de nombreux compromis, tels que la qualité de la réponse, la facilité d'implémentation et le nombre d'itérations nécessaires pour atteindre la convergence, pour n'en nommer que quelques-uns, qui doivent être pris en compte dans le processus d'optimisation.

2 Anexes

2.1 Gradient method algorithmes

```
1;

global ismydebugmode = false;

function debug_disp(msg)
  global ismydebugmode;
  if or(isdebugmode, ismydebugmode)
    disp(msg);
  endif
end

function alpha=aramijo_alpha_search(xk, dk, grk, f)
  debug_disp("Entering armijo alpha search");
  epsilon = 0.1;
  alpha = 1;
  fk = f(xk);
  debug_disp(["fk ", num2str(fk), " dk ", num2str(dk')
     ]);
  fka = f(xk + alpha*dk);
  k = 1;
  while fka > fk + epsilon*alpha*grk'*dk
    alpha = alpha/2;
    fka = f(xk + alpha*dk);
    debug_disp(["Iteration ", num2str(k), ";\talpha ",
       num2str(alpha), ";\tfka ", num2str(fka), ";\
       tcriteria ", num2str(fka - fk - epsilon*alpha*
       grk'*dk)]);
    k++;
  endwhile
  debug_disp(["Armijo search finalized with alpha: ",
     num2str(alpha)])
end

function alpha=parabolic_alpha_search(xk, dk, grk, f)
  debug_disp("Entering parabolic alpha search");
  alpha1 = 0;
  alpha3 = 1;

  f1 = f(xk);
  f3 = f(xk + alpha3*dk);
```

```
  debug_disp(["f1 ", num2str(f1), " f3 ", num2str(f3)])
     ;

  while f3 > f1
    alpha3 /= 2;
    f3 = f(xk+alpha3*dk);
  endwhile

  alpha2 = alpha3/2;
  f2 = f(xk + alpha2*dk);

  h1 = (f2-f1)/alpha2;
  h2 = (f3-f2)/alpha3;
  h3=(h2-h1)/(alpha3-alpha2);

  alpha0 = 1/2 * (alpha2 - h1/h3);
  f0 = f(xk + alpha0*dk);

  if f0 < f3
    alpha = alpha0;
  else
    alpha = alpha3;
  endif
end

function beta=beta_keanureeves_search(grk, grk1) %
   Fletcher-Reeves
  beta = grk1'*grk1/(grk'*grk);
end

function beta=beta_biere_search(grk, grk1) %
  beta = ((grk1 - grk)'*grk1)/(grk'*grk);
end

function [xmin, fmin, nbiter, iters, CONVCRIT]=steepest
   (x0, f, gr, varargin)
  % Default parameters
  tol = 0.01;
  iterlimit = 400;
  dkeps = 1e-10;
  flagx = false;
  flag_alpha = 1;
  flag_beta=0;
```

```
iter.x = [];
iter.f = [];
iter.alpha = [];
iter.beta = [];
iter.dk = [];
iter.gr = [];
iter.s = [];

DIM = rows(x0);

% Parse number of output arguments
if nargout==4
  flagx = true;
endif

% Parse input arguments
k = 1;
if nargin>3

  % First optional input
  if rem(nargin,2)==0
    method=varargin(1);
    k+=1;
  endif

  % Argument pairs to be parsed
  while k<length(varargin)
    s = cell2mat(varargin(k));
    k+=1;
    a = cell2mat(varargin(k));
    k+=1;
    switch (s)
      case 'alphamethod'
        alpha_search_method = a;
        if strcmp(alpha_search_method, 'aramijo')
          flag_alpha = 1;
        elseif strcmp(alpha_search_method , '
            parabolic')
          flag_alpha = 2;
        endif
      case 'betamethod'
        beta_search_method = a;
        if strcmp(beta_search_method, 'none')
          flag_beta = 0;
```

```
        elseif strcmp(beta_search_method, 'fletcher')
          flag_beta = 1;
        elseif strcmp(beta_search_method, 'ribière')
          flag_beta = 2;
        endif
      case 'tol'
        tol = a;
      case 'iterlimit'
        iterlimit = a;
      case 'dkeps'
        dkeps = a;
    endswitch
  endwhile
endif

% First iteration

xk = x0;
grk = gr(xk);
ngrk = norm(grk);
if ngrk < dkeps
  CONVCRIT = "Norm of the gradient too small";
endif
dk = -grk/ngrk;

if flag_alpha==1
  alphak = aramijo_alpha_search(xk, dk, grk, f);
else
  alphak = parabolic_alpha_search(xk, dk, grk, f);
endif

sk = alphak*dk;
xk1 = xk + sk;
fk1 = f(xk1);
fk = f(xk);

iter.x = [xk];
iter.f = [fk];
iter.alpha = [alphak];
iter.beta = [0];
iter.dk = [dk];
iter.gr = [grk];
iter.s = [sk];
iters = iter;
```

```
nbiter = 1;
precrel = norm(xk1 - xk)/norm(xk);

% Iteration loop
while precrel > tol && nbiter < iterlimit
  grk1 = gr(xk1);

  if flag_beta==0
    dk1 = -grk1;
    betak1 = 0;
  else
    if rem(nbiter, DIM)
      if flag_beta==1
        betak1 = beta_keanureeves_search(grk, grk1);
        grk = grk1;
      elseif flag_beta==2
        betak1 = beta_biere_search(grk, grk1);
        grk = grk1;
      end
      dk1 = -grk1 + betak1*dk;
    else
      dk1 = -grk1;
      betak1 = 0;
    endif
  endif

  ndk1 = norm(dk1);
  if ndk1 < dkeps
    CONVCRIT = "Step direction tolerance achieved";
    break;
  endif
  dk1 /= ndk1;

  if flag_alpha==1
    alphak1 = aramijo_alpha_search(xk1, dk1, grk1, f)
        ;
  elseif flag_alpha==2
    alphak1 = parabolic_alpha_search(xk1, dk1, grk1,
        f);
  endif

  sk1 = alphak1*dk1;
  xk2 = xk1 + sk1;
```

```
    precrel = norm(xk2 - xk1)/norm(xk1);
    if precrel <= tol
      CONVCRIT = "Convergence tolerance achieved";
    endif

    iter.x = xk1;
    iter.f = fk1;
    iter.alpha = alphak1;
    iter.beta = betak1;
    iter.dk = dk1;
    iter.gr = grk1;
    iter.s = sk1;
    iters(end+1) = iter;

    xk = xk1;
    xk1 = xk2;
    dk = dk1;
    fk1 = f(xk1);
    fk = f(xk);
    nbiter += 1;
    if nbiter >= iterlimit
      CONVCRIT = "Maximum number of iterations achieved
          ";
    endif
  endwhile

  xmin = xk2;
  fmin = f(xk2);

  iter.x = xk2;
  iter.f = fmin;
  iter.alpha = [];
  iter.beta = [];
  iter.dk = [];
  iter.gr = [];
  iter.s = [];
  iters(end+1) = iter;
end
```

2.2 Quasi-Newton methods algorithmes

```
1;

search_algorithms;
```

```
function [xmin, fmin, nbiter, iters, CONVCRIT]=minimize
    (x0, f, gr, varargin)
  % Default parameters
  tol = 0.01;
  iterlimit = 400;
  dkeps = 1e-10;
  flagx = false;
  flag_alpha = 1;
  flag_beta = 0;
  flag_dfp = false;
  flag_bfgs = false;

  iter.x = [];
  iter.f = [];
  iter.alpha = [];
  iter.beta = [];
  iter.dk = [];
  iter.gr = [];
  iter.s = [];
  iter.S = [];

  DIM = rows(x0);

  % Parse number of output arguments
  if nargout==4
    flagx = true;
  endif

  % Parse input arguments
  k = 1;
  if nargin>3

    % First optional input
    if rem(nargin,2)==0
      method=varargin(1);
      k+=1;
    endif

    % Argument pairs to be parsed
    while k<length(varargin)
      s = cell2mat(varargin(k));
      k+=1;
      a = cell2mat(varargin(k));
```

```
    k+=1;
    switch (s)
      case 'alphamethod'
        switch a;
          case 'aramijo'
            flag_alpha = 1;
          case 'armijo'
            flag_alpha = 1;
          case 'parabolic'
            flag_alpha = 2;
        endswitch
      case 'betamethod'
        switch a;
          case 'none'
            flag_beta = 0;
          case 'fletcher'
            flag_beta = 1;
          case 'ribière'
            flag_beta = 2;
        endswitch
      case 'tol'
        tol = a;
      case 'iterlimit'
        iterlimit = a;
      case 'dkeps'
        dkeps = a;
      case 'newtonmethod'
        switch a
          case 'dfp'
            flag_dfp = true;
          case 'bfgs'
            flag_bfgs = true;
        endswitch
    endswitch
  endwhile
endif

% First iteration

xk = x0;
grk = gr(xk);
ngrk = norm(grk);
if ngrk < dkeps
  CONVCRIT = "Norm of the gradient too small";
```

```
endif

dk = -grk;
Sk = eye(DIM);
dk = Sk*dk;

if flag_alpha==1
  alphak = aramijo_alpha_search(xk, dk, grk, f);
else
  alphak = parabolic_alpha_search(xk, dk, grk, f);
endif

sk = alphak*dk;
xk1 = xk + sk;
fk1 = f(xk1);
fk = f(xk);

iter.x = [xk];
iter.f = [fk];
iter.alpha = [alphak];
iter.beta = [0];
iter.dk = [dk];
iter.gr = [grk];
iter.s = [sk];
iter.S = [Sk];
iters = iter;

nbiter = 1;
precrel = norm(xk1 - xk)/norm(xk);

% Iteration loop
while precrel > tol && nbiter < iterlimit
  grk1 = gr(xk1);

  Sk1 = Sk;
  betak1 = 0;
  dk1 = -grk1;
  if flag_beta
    if rem(nbiter, DIM)
      if flag_beta==1
        betak1 = beta_keanureeves_search(grk, grk1);
      elseif flag_beta==2
        betak1 = beta_biere_search(grk, grk1);
      end
```

```
    dk1 = -grk1 + betak1*dk;
  endif
elseif flag_dfp
  gama_k = grk1 - grk;
  delta_k = xk1 - xk;
  vk = delta_k - Sk*gama_k;
  ak = 1/(vk'*gama_k);
  Ck = ak*vk*vk';

  Sk1 = Sk + Ck;
  dk1 = Sk1*dk1;
elseif flag_bfgs
  gama_k = grk1 - grk;
  delta_k = xk1 - xk;
  Ck = (1 + gama_k' * Sk * gama_k / (delta_k' *
     gama_k));
  Ck -= (delta_k * gama_k' * Sk + Sk * gama_k *
     delta_k') / (delta_k' * gama_k);

  Sk1 = Sk + Ck;
  dk1 = Sk1*dk1;
endif

ndk1 = norm(dk1);
if ndk1 < dkeps
  CONVCRIT = "Step direction tolerance achieved";
  break;
endif
dk1 /= ndk1;

if flag_alpha==1
  alphak1 = aramijo_alpha_search(xk1, dk1, grk1, f)
     ;
elseif flag_alpha==2
  alphak1 = parabolic_alpha_search(xk1, dk1, grk1,
     f);
endif

sk1 = alphak1*dk1;
xk2 = xk1 + sk1;
precrel = norm(xk2 - xk1)/norm(xk1);
if precrel <= tol
  CONVCRIT = "Convergence tolerance achieved";
endif
```

```
    iter.x = xk1;
    iter.f = fk1;
    iter.alpha = alphak1;
    iter.beta = betak1;
    iter.dk = dk1;
    iter.gr = grk1;
    iter.s = sk1;
    iter.S = Sk1;
    iters(end+1) = iter;

    xk = xk1;
    xk1 = xk2;
    dk = dk1;
    fk1 = f(xk1);
    fk = f(xk);
    grk = grk1;
    nbiter += 1;
    if nbiter >= iterlimit
      CONVCRIT = "Maximum number of iterations achieved
         ";
    endif
  endwhile

  xmin = xk2;
  fmin = f(xk2);

  iter.x = xk2;
  iter.f = fmin;
  iter.alpha = [];
  iter.beta = [];
  iter.dk = [];
  iter.gr = [];
  iter.s = [];
  iter.S = [];
  iters(end+1) = iter;
end
```

2.3 TP 1

2.3.1 Alpha Armijo

```
clear all;
close all;
clc;
```

```
addpath ".."
minimization_algorithms;
res_dir = "results";

% SETUP

global p = 100;

tol = 0.001;
alphamethod = 'aramijo';
betamethod = 'none'; % Gradient à pas optimal
iterlimit = 400;

x0=[0.5,0]';
s = 0.0005;

function f=banane(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  f = (x1 - 0.5).^2 + p*(x1.^2 - x2).^2;
end

function gr=gr(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  gr1 = 2*(x1-0.5) + 4*p*(x1.^2 - x2).*x1;
  gr2 = -2*p*(x1.^2 - x2);
  gr = [gr1; gr2];
end

% VISUALIZATION

##x1 = linspace(-2,2,100);
##x2 = linspace(-2,4,100);
x1 = linspace(0.3,0.6,1001);
x2 = linspace(-0.05,0.5,1001);
[X1, X2] = meshgrid(x1, x2);
X_shape = size(X1);

F = banane([X1(:)';X2(:)']);
F = reshape(F, X_shape);
```

```
##H = figure;
##hold on;
##colorbar;
##
##d0 = -gr(x0);
##quiver(x0(1), x0(2), s*d0(1), s*d0(2), 'linewidth',
    3);

% SOLUTION

[xmin, fmin, nbiter, iters, SC] = steepest(x0, @banane,
     @gr, 'tol', tol, 'alphamethod', alphamethod, '
    betamethod', betamethod, 'iterlimit', iterlimit);

H = figure;
hold on;
plot([iters.x](1,:), [iters.x](2,:), 'bo-');
labels = {};
for i = 1:length(iters)
  labels(end+1) = ['  x_{', num2str(i-1), '}'];
end
labels(2:end-1) = " ";
text([iters.x](1,:), [iters.x](2,:), labels, '
    horizontalalignment',"left", 'verticalalignment',"
    bottom", 'fontsize',35);
contour(x1, x2, F, logspace(log10(fmin),log10(max(max(F
    ))+1),15));
##axis equal;

##title(["x_f: ", num2str(xmin'), "\ncost_f: ", num2str
    (fmin), "\nNb-iter: ", num2str(nbiter)]);
grid on;
saveas(H, [res_dir, filesep, "tp1-p1-armijo-path"], "
    png");

disp(["xmin: ", num2str(xmin(1)),", ", num2str(xmin(2))
    ]);
disp(["fmin: ", num2str(fmin)]);
disp(["nbiter: ", num2str(nbiter)]);
```

2.3.2 Alpha Parabolic

```
clear all;
```

```
close all;
clc;

addpath ".."
minimization_algorithms;
res_dir = "results";

% SETUP

global p = 100;

tol = 0.001;
alphamethod = 'parabolic';
betamethod = 'none'; % Gradient à pas optimal
iterlimit = 400;

##x0 = [1,2]';
x0=[0.5,0]';
s = 0.0005;

function f=banane(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  f = (x1 - 0.5).^2 + p*(x1.^2 - x2).^2;
end

function gr=gr(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  gr1 = 2*(x1-0.5) + 4*p*(x1.^2 - x2).*x1;
  gr2 = -2*p*(x1.^2 - x2);
  gr = [gr1; gr2];
end

% VISUALIZATION

##x1 = linspace(-2,2,100);
##x2 = linspace(-2,4,100);
x1 = linspace(0.3,0.6,1001);
x2 = linspace(-0.05,0.5,1001);
[X1, X2] = meshgrid(x1, x2);
X_shape = size(X1);
```

```
F = banane([X1(:)';X2(:)']);
F = reshape(F, X_shape);

##H = figure;
##hold on;
##colorbar;
##
##d0 = -gr(x0);
##quiver(x0(1), x0(2), s*d0(1), s*d0(2), 'linewidth',
    3);

% SOLUTION

[xmin, fmin, nbiter, iters, SC] = steepest(x0, @banane,
     @gr, 'tol', tol, 'alphamethod', alphamethod, '
    betamethod', betamethod, 'iterlimit', iterlimit);

H = figure;
hold on;
plot([iters.x](1,:), [iters.x](2,:), 'bo-');
labels = {};
for i = 1:length(iters)
  labels(end+1) = ['  x_{', num2str(i-1), '}'];
end
labels(2:end-1) = " ";
text([iters.x](1,:), [iters.x](2,:), labels, '
    horizontalalignment',"left", 'verticalalignment',"
    bottom", 'fontsize',35);
contour(x1, x2, F, logspace(log10(fmin),log10(max(max(F
    ))+1),15));
##axis equal;

##title(["x_f: ", num2str(xmin'), "\ncost_f: ", num2str
    (fmin), "\nNb-iter: ", num2str(nbiter)]);
grid on;
saveas(H, [res_dir, filesep, "tp1-p1-parabolic-path"],
    "png");

disp(["xmin: ", num2str(xmin(1)),", ", num2str(xmin(2))
    ]);
disp(["fmin: ", num2str(fmin)]);
disp(["nbiter: ", num2str(nbiter)]);
```

2.3.3 Armijo Beta Fletcher

```
clear all;
close all;
clc;

addpath ".."
minimization_algorithms;
res_dir = "results";

% SETUP

global p = 100;

tol = 0.001;
alphamethod = 'aramijo';
betamethod = 'fletcher';
iterlimit = 400;

x0=[0.5,0]';
s = 0.0005;

function f=banane(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  f = (x1 - 0.5).^2 + p*(x1.^2 - x2).^2;
end

function gr=gr(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  gr1 = 2*(x1-0.5) + 4*p*(x1.^2 - x2).*x1;
  gr2 = -2*p*(x1.^2 - x2);
  gr = [gr1; gr2];
end

% VISUALIZATION

##x1 = linspace(-2,2,100);
##x2 = linspace(-2,4,100);
x1 = linspace(0.3,0.6,1001);
x2 = linspace(-0.05,0.5,1001);
```

```
[X1, X2] = meshgrid(x1, x2);
X_shape = size(X1);

F = banane([X1(:)';X2(:)']);
F = reshape(F, X_shape);

##H = figure;
##hold on;
##colorbar;
##
##d0 = -gr(x0);
##quiver(x0(1), x0(2), s*d0(1), s*d0(2), 'linewidth',
    3);

% SOLUTION

[xmin, fmin, nbiter, iters, SC] = steepest(x0, @banane,
     @gr, 'tol', tol, 'alphamethod', alphamethod, '
    betamethod', betamethod, 'iterlimit', iterlimit);

H = figure;
hold on;
plot([iters.x](1,:), [iters.x](2,:), 'bo-');
labels = {};
for i = 1:length(iters)
  labels(end+1) = ['  x_{', num2str(i-1), '}'];
end
labels(2:end-1) = " ";
text([iters.x](1,:), [iters.x](2,:), labels, '
    horizontalalignment',"left", 'verticalalignment',"
    bottom", 'fontsize',35);
contour(x1, x2, F, logspace(log10(fmin),log10(max(max(F
    ))+1),15));
##axis equal;

##title(["x_f: ", num2str(xmin'), "\ncost_f: ", num2str
    (fmin), "\nNb-iter: ", num2str(nbiter)]);
grid on;
saveas(H, [res_dir, filesep, "tp1-p1-aramijo-fletcher-
    path"], "png");

disp(["xmin: ", num2str(xmin(1)),", ", num2str(xmin(2))
    ]);
disp(["fmin: ", num2str(fmin)]);
```

```
disp(["nbiter: ", num2str(nbiter)]);
```

2.3.4 Armijo Beta Ribière

```
clear all;
close all;
clc;

addpath ".."
minimization_algorithms;
res_dir = "results";

% SETUP

global p = 100;

tol = 0.001;
alphamethod = 'aramijo';
betamethod = 'ribiere';
iterlimit = 400;

x0=[0.5,0]';
s = 0.0005;

function f=banane(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  f = (x1 - 0.5).^2 + p*(x1.^2 - x2).^2;
end

function gr=gr(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  gr1 = 2*(x1-0.5) + 4*p*(x1.^2 - x2).*x1;
  gr2 = -2*p*(x1.^2 - x2);
  gr = [gr1; gr2];
end

% VISUALIZATION

##x1 = linspace(-2,2,100);
##x2 = linspace(-2,4,100);
```

```
x1 = linspace(0.3,0.6,1001);
x2 = linspace(-0.05,0.5,1001);
[X1, X2] = meshgrid(x1, x2);
X_shape = size(X1);

F = banane([X1(:)';X2(:)']);
F = reshape(F, X_shape);

##H = figure;
##hold on;
##colorbar;
##
##d0 = -gr(x0);
##quiver(x0(1), x0(2), s*d0(1), s*d0(2), 'linewidth',
    3);

% SOLUTION

[xmin, fmin, nbiter, iters, SC] = steepest(x0, @banane,
     @gr, 'tol', tol, 'alphamethod', alphamethod, '
    betamethod', betamethod, 'iterlimit', iterlimit);

H = figure;
hold on;
plot([iters.x](1,:), [iters.x](2,:), 'bo-');
labels = {};
for i = 1:length(iters)
  labels(end+1) = ['␣␣x_{', num2str(i-1), '}'];
end
labels(2:end-1) = "␣";
text([iters.x](1,:), [iters.x](2,:), labels, '
    horizontalalignment',"left", 'verticalalignment',"
    bottom", 'fontsize',35);
contour(x1, x2, F, logspace(log10(fmin),log10(max(max(F
    ))+1),15));
##axis equal;

##title(["x_f: ", num2str(xmin'), "\ncost_f: ", num2str
    (fmin), "\nNb-iter: ", num2str(nbiter)]);
grid on;
saveas(H, [res_dir, filesep, "tp1-p1-aramijo-ribiere-
    path"], "png");

disp(["xmin:␣", num2str(xmin(1)),",␣", num2str(xmin(2))
```

```
    ]);
disp(["fmin: ", num2str(fmin)]);
disp(["nbiter: ", num2str(nbiter)]);
```

2.3.5 Parabolic Beta Fletcher

```
clear all;
close all;
clc;

addpath ".."
minimization_algorithms;
res_dir = "results";

% SETUP

global p = 100;

tol = 0.001;
alphamethod = 'parabolic';
betamethod = 'fletcher';
iterlimit = 400;

x0=[0.5,0]';
s = 0.0005;

function f=banane(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  f = (x1 - 0.5).^2 + p*(x1.^2 - x2).^2;
end

function gr=gr(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  gr1 = 2*(x1-0.5) + 4*p*(x1.^2 - x2).*x1;
  gr2 = -2*p*(x1.^2 - x2);
  gr = [gr1; gr2];
end

% VISUALIZATION
```

```
##x1 = linspace(-2,2,100);
##x2 = linspace(-2,4,100);
x1 = linspace(0.3,0.6,1001);
x2 = linspace(-0.05,0.5,1001);
[X1, X2] = meshgrid(x1, x2);
X_shape = size(X1);

F = banane([X1(:)';X2(:)']);
F = reshape(F, X_shape);

##H = figure;
##hold on;
##colorbar;
##
##d0 = -gr(x0);
##quiver(x0(1), x0(2), s*d0(1), s*d0(2), 'linewidth',
    3);

% SOLUTION

[xmin, fmin, nbiter, iters, SC] = steepest(x0, @banane,
    @gr, 'tol', tol, 'alphamethod', alphamethod, '
    betamethod', betamethod, 'iterlimit', iterlimit);

H = figure;
hold on;
plot([iters.x](1,:), [iters.x](2,:), 'bo-');
labels = {};
for i = 1:length(iters)
  labels(end+1) = ['  x_{', num2str(i-1), '}'];
end
labels(2:end-1) = " ";
text([iters.x](1,:), [iters.x](2,:), labels, '
    horizontalalignment',"left", 'verticalalignment',"
    bottom", 'fontsize',35);
contour(x1, x2, F, logspace(log10(fmin),log10(max(max(F
    ))+1),15));
##axis equal;

##title(["x_f: ", num2str(xmin'), "\ncost_f: ", num2str
    (fmin), "\nNb-iter: ", num2str(nbiter)]);
grid on;
saveas(H, [res_dir, filesep, "tp1-p1-parabolic-fletcher
    -path"], "png");
```

```
disp(["xmin: ", num2str(xmin(1)),", ", num2str(xmin(2))
    ]);
disp(["fmin: ", num2str(fmin)]);
disp(["nbiter: ", num2str(nbiter)]);
```

2.3.6 Parabolic Beta Ribière

```
clear all;
close all;
clc;

addpath ".."
minimization_algorithms;
res_dir = "results";

% SETUP

global p = 100;

tol = 0.001;
alphamethod = 'parabolic';
betamethod = 'ribiere';
iterlimit = 400;

x0=[0.5,0]';
s = 0.0005;

function f=banane(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  f = (x1 - 0.5).^2 + p*(x1.^2 - x2).^2;
end

function gr=gr(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  gr1 = 2*(x1-0.5) + 4*p*(x1.^2 - x2).*x1;
  gr2 = -2*p*(x1.^2 - x2);
  gr = [gr1; gr2];
end
```

```
% VISUALIZATION

##x1 = linspace(-2,2,100);
##x2 = linspace(-2,4,100);
x1 = linspace(0.3,0.6,1001);
x2 = linspace(-0.05,0.5,1001);
[X1, X2] = meshgrid(x1, x2);
X_shape = size(X1);

F = banane([X1(:)';X2(:)']);
F = reshape(F, X_shape);

##H = figure;
##hold on;
##colorbar;
##
##d0 = -gr(x0);
##quiver(x0(1), x0(2), s*d0(1), s*d0(2), 'linewidth',
    3);

% SOLUTION

[xmin, fmin, nbiter, iters, SC] = steepest(x0, @banane,
     @gr, 'tol', tol, 'alphamethod', alphamethod, '
    betamethod', betamethod, 'iterlimit', iterlimit);

H = figure;
hold on;
plot([iters.x](1,:), [iters.x](2,:), 'bo-');
labels = {};
for i = 1:length(iters)
  labels(end+1) = ['  x_{', num2str(i-1), '}'];
end
labels(2:end-1) = " ";
text([iters.x](1,:), [iters.x](2,:), labels, '
    horizontalalignment',"left", 'verticalalignment',"
    bottom", 'fontsize',35);
contour(x1, x2, F, logspace(log10(fmin),log10(max(max(F
    ))+1),15));
##axis equal;

##title(["x_f: ", num2str(xmin'), "\ncost_f: ", num2str
    (fmin), "\nNb-iter: ", num2str(nbiter)]);
grid on;
```

```
saveas(H, [res_dir, filesep, "tp1-p1-parabolic-ribiere-
    path"], "png");

disp(["xmin: ", num2str(xmin(1)),", ", num2str(xmin(2))
    ]);
disp(["fmin: ", num2str(fmin)]);
disp(["nbiter: ", num2str(nbiter)]);
```

2.3.7 Parabolic *DPF*

```
clear all;
close all;
clc;

addpath ".."
minimization_algorithms;
res_dir = "results";

% SETUP

global p = 100;

tol = 0.00000001;
alphamethod = 'parabolic';
betamethod = 'none';
newtonmethod = 'dfp';
iterlimit = 400;

x0=[0.5,0]';
s = 0.0005;

function f=banane(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  f = (x1 - 0.5).^2 + p*(x1.^2 - x2).^2;
end

function gr=gr(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  gr1 = 2*(x1-0.5) + 4*p*(x1.^2 - x2).*x1;
  gr2 = -2*p*(x1.^2 - x2);
```

```
  gr = [gr1; gr2];
end

% VISUALIZATION

##x1 = linspace(-2,2,100);
##x2 = linspace(-2,4,100);
x1 = linspace(0.3,0.6,1001);
x2 = linspace(-0.05,0.5,1001);
[X1, X2] = meshgrid(x1, x2);
X_shape = size(X1);

F = banane([X1(:)';X2(:)']);
F = reshape(F, X_shape);

##H = figure;
##hold on;
##colorbar;
##
##d0 = -gr(x0);
##quiver(x0(1), x0(2), s*d0(1), s*d0(2), 'linewidth',
   3);

% SOLUTION

[xmin, fmin, nbiter, iters, CONVCRIT] = minimize(x0,
   @banane, @gr, 'tol', tol, 'alphamethod', alphamethod
   , 'betamethod', betamethod, 'iterlimit', iterlimit,
   'newtonmethod', newtonmethod);

H = figure;
hold on;
plot([iters.x](1,:), [iters.x](2,:), 'bo-');
labels = {};
for i = 1:length(iters)
  labels(end+1) = ['  x_{', num2str(i-1), '}'];
end
labels(2:end-1) = " ";
text([iters.x](1,:), [iters.x](2,:), labels, '
   horizontalalignment',"left", 'verticalalignment',"
   bottom", 'fontsize',35);
contour(x1, x2, F, logspace(log10(fmin),log10(max(max(F
   ))+1),15));
##axis equal;
```

```
##title(["x_f: ", num2str(xmin'), "\ncost_f: ", num2str
    (fmin), "\nNb-iter: ", num2str(nbiter)]);
grid on;
saveas(H, [res_dir, filesep, "tp1-p1-dpf-parabolic-path
    "], "png");

disp(["xmin: ", num2str(xmin(1)),", ", num2str(xmin(2))
    ]);
disp(["fmin: ", num2str(fmin)]);
disp(["nbiter: ", num2str(nbiter)]);
disp(["Stop criteria: ", CONVCRIT]);
```

2.3.8 Parabolic *BFGS*

```
clear all;
close all;
clc;

addpath ".."
minimization_algorithms;
res_dir = "results";

% SETUP

global p = 100;

tol = 0.00000001;
alphamethod = 'parabolic';
betamethod = 'none';
newtonmethod = 'bfgs';
iterlimit = 400;

x0=[0.5,0]';
s = 0.0005;

function f=banane(x)
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  f = (x1 - 0.5).^2 + p*(x1.^2 - x2).^2;
end

function gr=gr(x)
```

```
  global p;
  x1 = x(1,:);
  x2 = x(2,:);
  gr1 = 2*(x1-0.5) + 4*p*(x1.^2 - x2).*x1;
  gr2 = -2*p*(x1.^2 - x2);
  gr = [gr1; gr2];
end

% VISUALIZATION

##x1 = linspace(-2,2,100);
##x2 = linspace(-2,4,100);
x1 = linspace(0.3,0.6,1001);
x2 = linspace(-0.05,0.5,1001);
[X1, X2] = meshgrid(x1, x2);
X_shape = size(X1);

F = banane([X1(:)';X2(:)']);
F = reshape(F, X_shape);

##H = figure;
##hold on;
##colorbar;
##
##d0 = -gr(x0);
##quiver(x0(1), x0(2), s*d0(1), s*d0(2), 'linewidth',
   3);

% SOLUTION

[xmin, fmin, nbiter, iters, CONVCRIT] = minimize(x0,
   @banane, @gr, 'tol', tol, 'alphamethod', alphamethod
   , 'betamethod', betamethod, 'iterlimit', iterlimit,
   'newtonmethod', newtonmethod);

H = figure;
hold on;
plot([iters.x](1,:), [iters.x](2,:), 'bo-');
labels = {};
for i = 1:length(iters)
  labels(end+1) = ['  x_{', num2str(i-1), '}'];
end
labels(2:end-1) = " ";
text([iters.x](1,:), [iters.x](2,:), labels, '
```

```
    horizontalalignment',"left", 'verticalalignment',"
    bottom", 'fontsize',35);
contour(x1, x2, F, logspace(log10(fmin),log10(max(max(F
    ))+1),15));
##axis equal;

##title(["x_f: ", num2str(xmin'), "\ncost_f: ", num2str
    (fmin), "\nNb-iter: ", num2str(nbiter)]);
grid on;
saveas(H, [res_dir, filesep, "tp1-p1-bfgs-parabolic-
    path"], "png");

disp(["xmin: ", num2str(xmin(1)),", ", num2str(xmin(2))
    ]);
disp(["fmin: ", num2str(fmin)]);
disp(["nbiter: ", num2str(nbiter)]);
disp(["Stop criteria: ", CONVCRIT]);
```

2.4 TP 2

2.4.1 Problème 0

```
clear all;
close all;
clc;

addpath ".."
minimization_algorithms;
res_dir = "results";

tol = 0.00001;
alphamethod = 'aramijo';
betamethod = 'fletcher';
iterlimit = 400;

x0=[0.5];

global eta = 0.01;

function f0=f0(x)
  f0 = 1 + x + 1/3*x.*x.*x;
end

function p=p(x)
  p = min([x; zeros(1,length(x))], [], 1);
```

```
  p .*= p;
end

function f=f(x)
  global eta;
  f = f0(x) + 1/eta*p(x);
end

function g=grf0(x)
  g = 1 + x.*x;
end

function g=grp(x)
  g = 2*x;
  g(x>0) = 0;
end

function g=gr(x)
  global eta;
  g = grf0(x) + 1/eta*grp(x);
end

[xmin, fmin, nbiter, iters, CONVCRIT] = steepest(x0, @f
    , @gr, 'tol', tol, 'alphamethod', alphamethod, '
    betamethod', betamethod, 'iterlimit', iterlimit);

disp(["xmin: ", num2str(xmin)]);
disp(["fmin: ", num2str(fmin)]);
disp(["nbiter: ", num2str(nbiter)]);

x = linspace(-0.3,1,1001);

H = figure;
hold on;
plot([iters.x], [iters.f], '*', 'markersize', 10);
plot(x, f(x));

labels = {};
for i = 0:nbiter
  labels(end+1) = num2str(i);
end

labels(2:end-1) = " ";
text([iters.x], [iters.f], labels, 'horizontalalignment
```

```
    ',"left", 'verticalalignment',"bottom", 'fontsize'
    ,15);

grid on;
saveas(H, [res_dir, filesep, "p0-path"], "png");
```

2.4.2 Problème 1

```
clear all;
close all;
clc;

addpath ".."
minimization_algorithms;
res_dir = "results";

tol = 0.001;
alphamethod = 'parabolic';
betamethod = 'fletcher';
iterlimit = 400;

x0=[-0.35; -0.31];

global eta = 0.01;

function f1=f1(x)
  f1 = 2*x(1,:).^2 + 3*x(1,:).*x(2,:) + 2*x(2,:).^2;
end

function p=p(x)
  x(1,:) += 0.5;
  x(2,:) += 0.5;
  x(1,x(1,:)<0) = 0;
  x(2,x(2,:)<0) = 0;
  p = sum(x.*x, 1);
end

function f=f(x)
  global eta;
  f = f1(x) + 1/eta*p(x);
end

function g=grf1(x)
  g1 = 4*x(1,:) + 3*x(2,:);
```

```
  g2 = 3*x(1,:) + 4*x(2,:);
  g = [g1; g2];
end

function g=grp(x)
  x(1,:) += 0.5;
  x(2,:) += 0.5;
  x(1,x(1,:)<0) = 0;
  x(2,x(2,:)<0) = 0;
  g1 = 2*x(1,:);
  g2 = 2*x(2,:);
  g = [g1; g2];
end

function g=gr(x)
  global eta;
  g = grf1(x) + 1/eta*grp(x);
end

[xmin, fmin, nbiter, iters, CONVCRIT] = steepest(x0, @f
    , @gr, 'tol', tol, 'alphamethod', alphamethod, '
    betamethod', betamethod, 'iterlimit', iterlimit);

disp(["xmin: ", num2str(xmin')]);
disp(["fmin: ", num2str(fmin)]);
disp(["nbiter: ", num2str(nbiter)]);
disp(["Stop criteria: ", CONVCRIT]);

x1 = linspace(-1,-0.3,101);
x2 = linspace(-1,-0.3,101);

[XX1, XX2] = meshgrid(x1, x2);
xx1 = XX1(:)';
xx2 = XX2(:)';
F = f([xx1;xx2]);
F = reshape(F, size(XX1));

minF = min(min(F));
maxF = max(max(F));

% 2 dimensional plot

H = figure;
hold on;
```

```
plot([iters.x](1,:), [iters.x](2,:), 'bo-');
labels = {};
for i = 0:nbiter
  labels(end+1) = ['  x_{', num2str(i), '}'];
end
labels(2:end-1) = " ";
text([iters.x](1,:), [iters.x](2,:), labels, '
   horizontalalignment',"left", 'verticalalignment',"
   bottom", 'fontsize',15);

contour(x1, x2, F, logspace(-4,log10(maxF-minF),25)+
   minF);
colorbar;

x0 = [-0.75,-0.75]';
gr0 = gr(x0);
s = 0.0005;
quiver(x0(1), x0(2), s*gr0(1), s*gr0(2), 'r')
##axis equal;

grid on;
saveas(H, [res_dir, filesep, "p1-contour-path"], "png")
   ;

% 3 dimensional plot

H = figure;
hold on;
surface(x1, x2, F, 'linestyle', 'none');
plot3([iters.x](1,:), [iters.x](2,:), [iters.f](1,:), '
   bo-');

labels = {};
for i = 0:nbiter
  labels(end+1) = ['  x_{', num2str(i), '}'];
end

labels(2:end-1) = " ";
text([iters.x](1,:), [iters.x](2,:), [iters.f]+0.5,
   labels, 'horizontalalignment',"right", '
   verticalalignment',"bottom", 'fontsize',15, 'color',
    'r');
colorbar;
```

```
grid on;
view(20,45);
saveas(H, [res_dir, filesep, "p1-3d-path"], "png");
```

2.4.3 Problème 2

```
clear all;
close all;
clc;

addpath ".."
minimization_algorithms;
res_dir = "results";

tol = 0.001;
alphamethod = 'parabolic';
betamethod = 'fletcher';
iterlimit = 400;

x0=[0.5];

global eta = 0.01;

function f2=f2(x)
  f2 = x.*x + 10*sin(x);
end

function p=p(x)
  xa = x - 2;
  xb = x - 10;
  xa(xa>0) = 0;
  xb(xb<0) = 0;
  p = xa.^2+xb.^2;
end

function f=f(x)
  global eta;
  f = f2(x) + 1/eta*p(x);
end

function g=grf1(x)
  g = 2*x + 10*cos(x);
end
```

```
function g=grp(x)
  xa = x - 2;
  xb = x - 10;
  xa(xa>0) = 0;
  xb(xb<0) = 0;
  g = 2*xa+2*xb;
end

function g=gr(x)
  global eta;
  g = grf1(x) + 1/eta*grp(x);
end

[xmin, fmin, nbiter, iters, CONVCRIT] = steepest(x0, @f
    , @gr, 'tol', tol, 'alphamethod', alphamethod, '
    betamethod', betamethod, 'iterlimit', iterlimit);

disp(["xmin: ", num2str(xmin')]);
disp(["fmin: ", num2str(fmin)]);
disp(["nbiter: ", num2str(nbiter)]);
disp(["Stop criteria: ", CONVCRIT]);

x = linspace(0,12,101);

% 2 dimensional plot

H = figure;
hold on;
plot(x, f(x), 'displayname', "f");
plot([iters.x], [iters.f], 'bo-');

labels = {};
for i = 0:nbiter
  labels(end+1) = ['  x_{', num2str(i), '}'];
end

labels(2:end-1) = " ";
text([iters.x], [iters.f], labels, 'horizontalalignment
    ',"left", 'verticalalignment',"bottom", 'fontsize'
    ,15);

grid on;
saveas(H, [res_dir, filesep, "p2-path"], "png");
```

2.4.4 Problème 3

```
clear all;
close all;
clc;

addpath ".."
minimization_algorithms;

tol = 1e-6;
alphamethod = 'parabolic';
betamethod = 'fletcher';
iterlimit = 1000;

x0=[2, 1, 2]';

global eta = 0.01;
global A = [5 3 1; 3 2 1; 1 1 4];

function f3=f3(x)
  global A;
  f3 = x'*A*x;
end

function p=p(x)
  x1 = x(1);
  x2 = x(2);
  x3 = x(3);

  p1 = (x1 + x2 - x3).^2;
  p1(x1 + x2 < x3) = 0;

  p2 = (3*x1 - 2*x2 - x3).^2;

  p = p1+p2;
end

function f=f(x)
  global eta;
  f = f3(x) + 1/eta*p(x);
end

function g=grf3(x)
  global A;
```

```
  gx1 = A(1,:)*x + x'*A(:,1);
  gx2 = A(2,:)*x + x'*A(:,2);
  gx3 = A(3,:)*x + x'*A(:,3);
  g = [gx1, gx2, gx3]';
end

function g=grp(x)
  x1 = x(1);
  x2 = x(2);
  x3 = x(3);

  g1 = 2*(x1 + x2 - x3)*[1 1 -1]';
  g1(x1 + x2 < x3) = 0;

  g2 = 2*(3*x1 - 2*x2 - x3)*[3 -2 -1]';

  g = g1 + g2;
end

function g=gr(x)
  global eta;
  g = grf3(x) + 1/eta*grp(x);
end

[xmin, fmin, nbiter, iters, SC] = steepest(x0, @f, @gr,
    'tol', tol, 'alphamethod', alphamethod, 'betamethod
   ', betamethod, 'iterlimit', iterlimit);

disp(["xmin: ", num2str(xmin')]);
disp(["fmin: ", num2str(fmin)]);
disp(["nbiter: ", num2str(nbiter)]);
disp(["Stop criteria: ", SC]);
```

2.4.5 Problème 4

```
clear all;
close all;
clc;

addpath ".."
minimization_algorithms;

tol = 0.0001;
alphamethod = 'parabolic';
```

```
betamethod = 'fletcher';
iterlimit = 1000;

x0=[10 -1 0.1 4]';

global eta = 0.01;
global xp = [1 1 1 1]';
global A = [3 1 0 2];
global b = 1;

function f2=f2(x)
  global xp;
  f2 = 1/2*(x-xp)'*(x-xp);
end

function p=p(x)
  global A;
  global b;
  p = (A*x - b).^2;
end

function f=f(x)
  global eta;
  f = f2(x) + 1/eta*p(x);
end

function g=grf1(x)
  global xp;
  g = x - xp;
end

function g=grp(x)
  global A;
  global b;
  g = 2*(A*x - b)*A';
end

function g=gr(x)
  global eta;
  g = grf1(x) + 1/eta*grp(x);
end

[xmin, fmin, nbiter, iters, SC] = steepest(x0, @f, @gr,
    'tol', tol, 'alphamethod', alphamethod, 'betamethod
```

```
    ', betamethod, 'iterlimit', iterlimit);

disp(["xmin: ", num2str(xmin')]);
disp(["fmin: ", num2str(fmin)]);
disp(["nbiter: ", num2str(nbiter)]);
disp(["Stop criteria: ", SC]);

xan = xp+A'*inv(A*A')*(b-A*xp);
fan = 1/2*(xan-xp)'*(xan-xp);
dann = sqrt(sum((xan-xmin).^2));

disp(["Analytical result: ", num2str(xan')]);
disp(["Analytical distance: ", num2str(fan)]);
disp(["Distance between analytical and numerical
    results: ", num2str(dann)]);
```